Optimizing our App

- Code splitting
- Lazy loading
- Performant web scalable Application
- suspense
- chunking
- Custom Hooks

→ we know that Hooks are nothing but javascript function.

→ we can create JS function for different functionality as in same way we can create Hooks as well by ourselves.

→ They are known as custom Hooks.

Qs when, how and why one should build ~~their~~ our Hook?

Qs Why custom Hooks? (same reason as how we create fxn)

→ Reusability
→ Readability
→ Separation of concerns (different Hooks for different things)
→ Maintainability (it is easy to debug).
→ makes our code more testable (one can check each hooks individually).
→ Abstraction of logic

How to create Hooks !

→ same as the other components

→ use prefix (use) for it.

Difference b/w functional components and custom Hooks

→ It doesn't return JSX, it only offloads some logic from a component and put it hooks and in that a way make a component more robust, maintainable and readable,

→ Making a feature which shows whether you are online or not (custom Hook example).

→ online event can be used to track so

e.g) addEventListener('online', (event) => { });

Making useOnline() ; custom Hook

```
const useOnline = () => {
const [isOnline, setIsOnline] = useState(true);

useEffect(() => {
window.addEventListener('online', () => { setIsOnline(true); });
window.addEventListener('offline', () => { setIsOnline(false); });
}, []);
return isOnline;
};
export default useOnline;
```

→ cleaning the cache

wherever you add up new event-listener you also need to clean it up.

because

i) whenever that condition becomes true, everytime a new event-listener will be getting called (here it is not, since empty dependency array will make sure to call only once).

ii) whenever we move to other component where that event listener is not needed then why to take extra load if one can remove it.

(clear the mess you have created) (unmount cycle).

Tasks create a HOOK to use local storage.

→ Basic functionality (main fxn) of all bundlers:

- To bundle things up, it creates a single JS file for all the code.
- Any React web APP may contain numerous (hundreds or thousands) of components.
- If bundler binds all those components inside one file it will be very less performant web APP.

→ When we have to create production ready App ~~they~~ (it can't be created by just one bundle.

solution

↓ Introducing code splitting, chunking / bundling, lazy loading (all are synonyms) ↓ on demand loading, Dynamic import.

Q1 Is bundling good or bad?

→ Its good within certain limits.

→ we should build logical bundles.

→ Different builds get builded upon different use cases (it is all dependent on wisdom of developer).

(System Design).

Suppose a website, let's say MMT has different features (or services) such as flights, hotels, homestays, holiday packages, Trains, buses, cabs, forex, charter flights and each service has build of 100 components, so it doesn't make sense to load all the components if we are visiting a particular service (user will need functionality of that service only).

→ Whenever our home page loads, it just loads the components for our main service (here flights).

→ If we move to other services, then components of that service will be loaded.

→ It is known as on demand loading (all synonyms).

eg) In this we will not import components as we are used to, we will use lazy loading and import the components dynamically.

```
const Istamart = lazy(() => import("./components/Istamart"));
```

(dynamic import; it is basically as Promise; path given by React library)

→ this component will not be added to the initial bundle when we move to that component then it will slip in.

→ Above code will give error (only for first time)

why? → since it is different bundle altogether, it takes some time to load the script and meanwhile React tries to render it (which is actually not present). In this situation, React suspends the rendering. (operation)

→ for avoiding this error, React gives something called suspense

→ since that dynamic import is promise, if we put that component inside suspense, they react will wait until that promise gets resolved.

→ prop named fallback

<Suspense fallback={<Dinner/>}>

contains those things which we want user to see until the comp promise gets resolved.

NOTE: Never ever dynamically import a component inside other component.

(Do lazy loading along with other imports) on the top)